# Enkapsulasi

Menyembunyikan elemen dari penggunaan sebuah class dapat dilakukan dengan pembuatan anggota yang ingin Anda sembunyikan secara private.

Contoh berikut menyembunyikan field *secret*.

Catatan bahwa field ini tidak langsung diakses oleh program lain menggunakan method getter dan setter.

```
class Encapsulation {
     private int secret; //field tersembunyi
     public boolean setSecret(int secret) {
          if (secret < 1 || secret > 100) {
               return false;
     }
          this.secret = secret;
          return true;
     }
     public getSecret() {
     return secret;
     }
}
```

# Pewarisan

Untuk membuat class anak atau subclass berdasarkan class yang telah ada, kita gunakan kata kunci ***extends*** dalam mendeklarasikan class.
Sebuah class hanya dapat melakukan extension terhadap satu parent class.

Sebagai contoh, class *Point* di bawah ini adalah superclass dari class *ColoredPoint.*

```
import java.awt.*;
class Point {
     int x;
     int y;
}

class ColoredPoint extends Point {
     Color color;
}
```

## Polimorfisme

Suppose Rectangle derives from Quadrilateral
- Rectangle more specific than Quadrilateral
- Any operation on Quadrilateral can be done on Rectangle (i.e., perimeter, area)

Suppose designing Video Game
- Superclass SpaceObject
  - Subclasses Martian, SpaceShip, LaserBeam
  - Contains method draw
- To refresh screen
  - Send draw message to each object
  - Same message has “many forms” of results
- Easy to add class Mercurian
  - Extends SpaceObject
  - Provides its own implementation of draw
- Programmer does not need to change code
  - Calls draw regardless of object’s type
  - Mercurian objects “plug right in”

## Abstract Class

- Abstract classes
  - Are superclasses (called abstract superclasses)
  - Cannot be instantiated
  - Incomplete
    - ~ subclasses fill in "missing pieces"
- Abstract classes not required, but reduce client code dependencies
- To make a class abstract
  - Declare with keyword abstract
  - Contain one or more *abstract methods*

```
public abstract void draw();
```

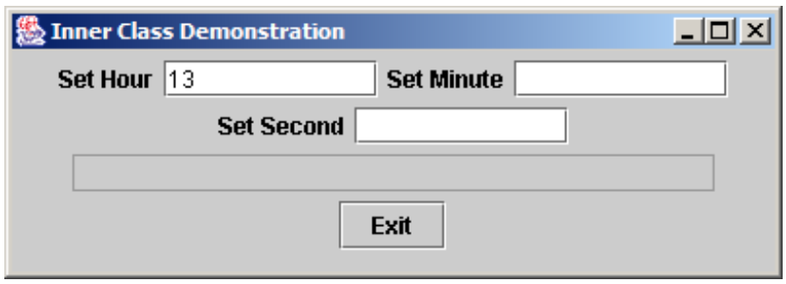
  - Abstract methods
    - ~ No implementation, must be overridden
- Application example
  - Abstract class Shape
    - ~ Declares draw as abstract method
  - Circle, Triangle, Rectangle extends Shape
    - ~ Each must implement draw
  - Each object can draw itself

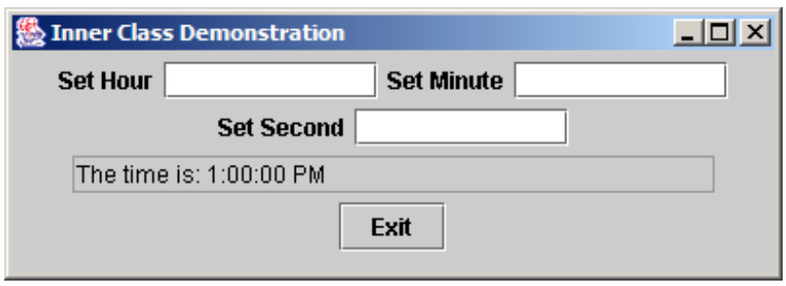
```
// Fig. 10.21: Time.java
// Time class declaration with set and get methods.
import java.text.DecimalFormat;

public class Time {
   private int hour;     // 0 - 23
   private int minute;   // 0 - 59
   private int second;   // 0 - 59

 // one formatting object to share in toString and toUniversalString
 private static DecimalFormat twoDigits = new DecimalFormat( "00" );

 // Time constructor initializes each instance variable to zero;
 // ensures that Time object starts in a consistent state
 public Time()
 {
    this( 0, 0, 0 ); // invoke Time constructor with three arguments
 }

 // Time constructor: hour supplied, minute and second defaulted to 0
 public Time( int h )
 {
    this( h, 0, 0 ); // invoke Time constructor with three arguments
 }
 // Time constructor: hour and minute supplied, second defaulted to 0
 public Time( int h, int m )
 {
    this( h, m, 0 ); // invoke Time constructor with three arguments
 }

 // Time constructor: hour, minute and second supplied
 public Time( int h, int m, int s )
 {
    setTime( h, m, s );
 }

 // Time constructor: another Time3 object supplied
 public Time( Time time )
 {
    // invoke Time constructor with three arguments
    this( time.getHour(), time.getMinute(), time.getSecond() );
 }

 // Set Methods
 // set a new time value using universal time; perform
 // validity checks on data; set invalid values to zero
 public void setTime( int h, int m, int s )
 {
    setHour( h );   // set the hour
    setMinute( m ); // set the minute
    setSecond( s ); // set the second
 }

 // validate and set hour
 public void setHour( int h )
 {
    hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
 }

 // validate and set minute
 public void setMinute( int m )
 {
    minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
 }

 // validate and set second
 public void setSecond( int s )
 {
    second = ( ( s >= 0 && s < 60 ) ? s : 0 );
 }

 // Get Methods
```

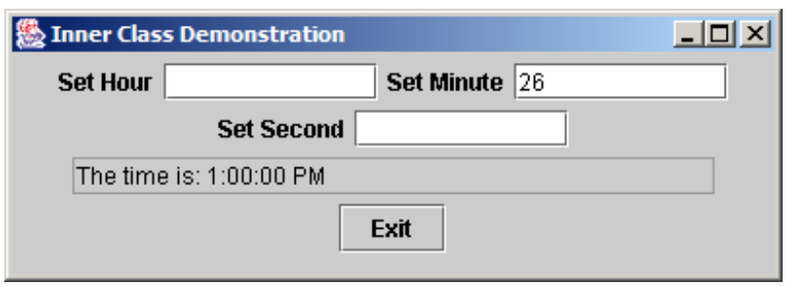
```
   // get hour value
   public int getHour()
   {
      return hour;
   }

   // get minute value
   public int getMinute()
   {
      return minute;
   }

   // get second value
   public int getSecond()
   {
      return second;
   }

   // convert to String in universal-time format
   public String toUniversalString()
   {
      return twoDigits.format( getHour() ) + ":" +
         twoDigits.format( getMinute() ) + ":" +
         twoDigits.format( getSecond() );
   }

// convert to String in standard-time format
public String toString()
{
   return ( ( getHour() == 12 || getHour() == 0 ) ?
      12 : getHour() % 12 ) + ":" + twoDigits.format( getMinute() ) +
      ":" + twoDigits.format( getSecond() ) +
      ( getHour() < 12 ? " AM" : " PM" );
}

} // end class Time
```

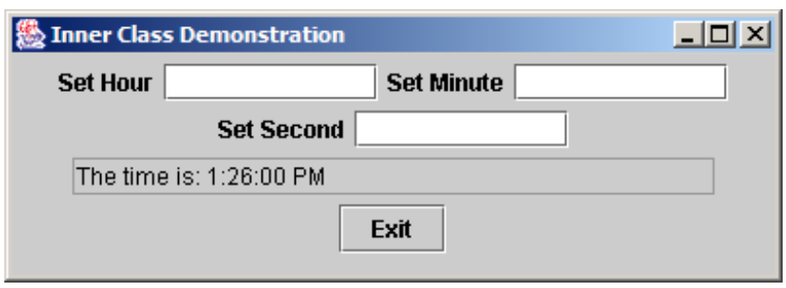
```
// Fig. 10.22: TimeTestWindow.java
     // Inner class declarations used to create event handlers.
     import java.awt.*;
     import java.awt.event.*;
     import javax.swing.*;

     public class TimeTestWindow extends JFrame
     {
        private Time time;
        private JLabel hourLabel, minuteLabel, secondLabel;
       private JTextField hourField, minuteField, secondField, displayField;
       private JButton exitButton;

       // set up GUI
       public TimeTestWindow()
       {
          // call JFrame constructor to set title bar string
          super( "Inner Class Demonstration" );

          time = new Time();  // create Time object

          // use inherited method getContentPane to get window's content pane
          Container container = getContentPane();
          container.setLayout( new FlowLayout() );  // change layout

          // set up hourLabel and hourField
          hourLabel = new JLabel( "Set Hour" );
          hourField = new JTextField( 10 );
          container.add( hourLabel );
          container.add( hourField );

          // set up minuteLabel and minuteField
          minuteLabel = new JLabel( "Set Minute" );
          minuteField = new JTextField( 10 );
          container.add( minuteLabel );
          container.add( minuteField );

          // set up secondLabel and secondField
          secondLabel = new JLabel( "Set Second" );
          secondField = new JTextField( 10 );
          container.add( secondLabel );
          container.add( secondField );

          // set up displayField
          displayField = new JTextField( 30 );
          displayField.setEditable( false );
          container.add( displayField );

          // set up exitButton
          exitButton = new JButton( "Exit" );
          container.add( exitButton );

          // create an instance of inner class ActionEventHandler
          ActionEventHandler handler = new ActionEventHandler();

          // register event handlers; the object referenced by handler
          // is the ActionListener, which contains method actionPerformed
          // that will be called to handle action events generated by
          // hourField, minuteField, secondField and exitButton
          hourField.addActionListener( handler );
          minuteField.addActionListener( handler );
          secondField.addActionListener( handler );
          exitButton.addActionListener( handler );

       } // end constructor

       // display time in displayField
       public void displayTime()
       {
          displayField.setText( "The time is: " + time );
       }
```

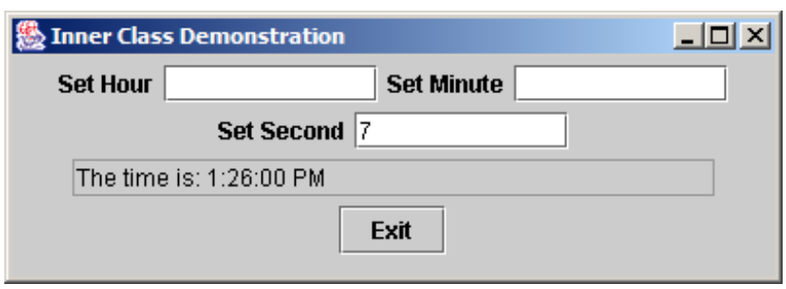
```
   // launch application: create, size and display TimeTestWindow;
   // when main terminates, program continues execution because a
   // window is displayed by the statements in main
   public static void main( String args[] )
   {
      TimeTestWindow window = new TimeTestWindow();

      window.setSize( 400, 140 );
      window.setVisible( true );

   } // end main

   // inner class declaration for handling JTextField and JButton events
   private class ActionEventHandler implements ActionListener
   {

      // method to handle action events
      public void actionPerformed( ActionEvent event )
      {
         // user pressed exitButton
         if ( event.getSource() == exitButton )
            System.exit( 0 );   // terminate the application

         // user pressed Enter key in hourField
         else if ( event.getSource() == hourField ) {
            time.setHour( Integer.parseInt(
               event.getActionCommand() ) );
            hourField.setText( "" );
         }

       // user pressed Enter key in minuteField
       else if ( event.getSource() == minuteField ) {
          time.setMinute( Integer.parseInt(
             event.getActionCommand() ) );
          minuteField.setText( "" );
       }
      }
   }
}
```

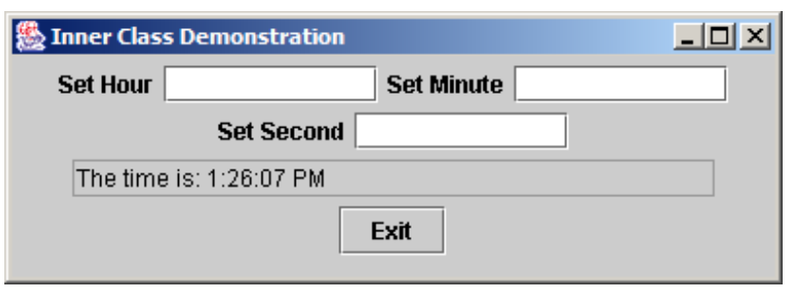

Inner Class Demonstration
Set Hour 13
Set Minute
Set Second
Exit


Inner Class Demonstration
Set Hour
Set Minute
Set Second
The time is: 1:00:00 PM
Exit


Inner Class Demonstration
Set Hour
Set Minute 26
Set Second
The time is: 1:00:00 PM
Exit


Inner Class Demonstration
Set Hour
Set Minute
Set Second
The time is: 1:26:00 PM
Exit


Inner Class Demonstration
Set Hour
Set Minute
Set Second 7
The time is: 1:26:00 PM
Exit


Inner Class Demonstration
Set Hour
Set Minute
Set Second
The time is: 1:26:07 PM
Exit